# Principles and experimentations of self-organizing embedded agents allowing learning from demonstration in ambient robotics

Nicolas Verstaevel [a,b,*], Christine Régis [a], Marie-Pierre Gleizes [a], Fabrice Robert [b]

[a] IRIT, Université Paul Sabatier, 118 rte de Narbonne, 31062 Toulouse Cedex, France
[b] Sogeti High Tech, 3 Chemin Laporte, 31300 Toulouse, France

A B S T R A C T

Ambient systems are populated by many heterogeneous devices to provide adequate services to their users. The adaptation of an ambient system to the specific needs of its users is a challenging task. Because human–system interaction has to be as natural as possible, we propose an approach based on Learning from Demonstration (LfD). LfD is an interesting approach to generalize what has been observed during the demonstration to similar situations. However, using LfD in ambient systems needs adaptivity of the learning technique. We present ALEX, a multi-agent system able to dynamically learn and reuse contexts from demonstrations performed by a tutor. The results of the experiments performed on both a real and a virtual robot show interesting properties of our technology for ambient applications.

*Keywords:*
Adaptive agents
Cooperative multi-agent systems
Robotics in ambient systems
Machine learning
Context-awareness

## 1. Introduction

Once confined in a science of control in structured environments, researches on robotics are now considering the integration of robotic devices in the real world for applications where tasks are diverse, complex and evolutive [1]. Service robotic differs from its industrial version by the interest in providing services to humans. Consequently, research now consider the use of robotic devices in ambient applications [2] and the term *ambient robotics* directly refers to the usage of robotic components in ambient systems. Ambient systems are characterized by their high dynamic and their complexity. Many heterogeneous robotic devices can appear and disappear along the system life-cycle and interact opportunistically together. According to the definition of Russell and Norvig [3], the environments of ambient systems are:

- **Inaccessible**: each device composing the system has a partial observation of the environment.
- **Continuous**: considering applications in the real world, the number of observations and actions is not discrete.
- **Non-deterministic**: consequences of performed actions in the real world could not be determined in advance with certainty.
- **Dynamic**: system's actions, user activity, appearance and disappearance of devices may change the environment.

Consequently, designing an *ad hoc* controller of a robotic device in an ambient system is a complex task that requires a lot of knowledge. This complexity is increased if we take into account that users have multiple, specific and often changing needs. Providing to those devices the ability to learn and adapt to users' needs is then a particularly challenging task [4]. To be as natural as possible, such learning ability needs to rest on a process that does not require any kind of technical knowledge for users (i). Furthermore, it needs both genericity, to be applicable on any kind of devices with any kind of users, and openness properties to deal with the appearance and disappearance of devices. The genericity and openness properties require then using agnostic learning techniques that makes as

* Corresponding author at: IRIT, Université Paul Sabatier, 118 rte de Narbonne, 31062 Toulouse Cedex, France.
*E-mail address:* nicolas.verstaevel@irit.fr (N. Verstaevel).

few assumptions as possible [5] (ii). To deal with (i) and (ii), we propose to use Learning from Demonstration (LfD), a paradigm to dynamically learn new behaviours. The paper is organized as follows: first, we present the problems and challenges of LfD in ambient systems. Then, we present ALEX our solution to handle this challenge. Two experiments are then proposed to illustrate ALEX's behaviour. At last, the conclusion discusses perspectives and future works.

## 2. Learning from demonstration

### 2.1. General principle

Learning from Demonstration, also named "Imitation Learning" or "Programming by Demonstration", is a paradigm mainly studied in the robotic field that allows systems to self-discover new behaviours [6]. It takes inspiration from the natural tendency of some animal species and humans to learn from the imitation of their congeners. The main idea is that an appropriate controller for a robotic device can be learnt from the observation of the performance of another entity (virtual or human) named as the *tutor*. The tutor can interact with the system to explicit the desired behaviour through the natural process of demonstration. A demonstration is then a set of successive actions performed by the tutor in a particular context. The learning system has to produce a mapping function correlating observations of the environment and tutor's actions to its own actions. The main advantage of such technique is that it needs no explicit programming or knowledge on the system. It only observes tutor's actions and current system context to learn a control policy and can be used by end-users without technical skills.

The paradigm has been used on a wide range of applications such as autonomous car following [7], robot trajectory learning [8], robot navigation in complex unstructured terrain [9] or haptic guidance of bimanual surgical tasks [10]. Recent surveys [11,6] propose an overview of the LfD field illustrating a wide variety of applications. Our interest is not to focus on one particular application. On the contrary, we want to deal with any kind of ambient robotic system. This section ends with a study of the usage of the LfD paradigm in the context of ambient robotics.

### 2.2. Problem formalization

Billing and Hellström [12] propose a complete LfD formalization. Here, we propose to introduce the fundamental concepts of LfD and discuss its usage in ambient robotics.

LfD is a problem of imitation as an entity tries to produce a behaviour similar to another entity. A tutor evolving in a world realizes an observation $\Omega$ of this world. The tutor can perform a set of actions $A$ ($A$ could be empty) and follows a policy $\pi_{tutor}$ (1) that associates to any world state a particular action. It is supposed that (1) is the optimal policy to satisfy the tutor.

- $\pi_{tutor} : \Omega \rightarrow a \in A.$ (1)

The learner disposes of a set of observations $O$ (named observation space) on the world and its own set of actions $B$. The learner follows another policy $\pi_{learner}$ (2) in order to produce a behaviour similar to the observed one.

- $\pi_{learner} : O \rightarrow b \in B,\ \pi_{learner} \equiv \pi_{tutor}.$ (2)

In most cases, the tutor and the system, while evolving in the same world (which can be a virtual world or the real world), have a different observation of the world. It is particularly true in real world problems with human tutors where the world is observed by the system through sensors whereas the human observes it through its own senses. It results in a problem of perception equivalence [13]. The tutor demonstrating a particular behaviour can observe modifications of the world that the learner cannot perceive. However, equivalences of perception can be found by the system. For example, a user is cold and turns the heating on. Observing through sensors that a user is cold is complex, but observing the current temperature, wind and humidity levels is easily feasible. A learner can make a correlation between the current situation described by sensors and the action of turning the heating on and it can learn that it is necessary to turn the heating on when a similar situation occurs. The learner has to find correlations between its own observations and the performance of an action by the tutor. This raises the challenging question of the possible lack of perception. A tutor can perform a demonstration dependent of a phenomenon that the learner cannot observe. In this paper, we consider that the learner has sufficient observations to perform the task. Nevertheless, some clues to handle this problem are proposed in Section 5 as perspectives. The problem to tackle is then how to interpret those observations to construct a control policy enabling the learner to produce an imitative behaviour.

### 2.3. Lfd and ambient systems

Ambient systems are rich of interaction possibilities for users. We propose to exploit the inherent interactivity of ambient systems in order to learn and adapt from users' activity. LfD then appears to be a suitable paradigm to learn and generalize recurrent activities from the observation of users' activity. Therefore, users can be seen as system's tutors. Each user's action on a device is then seen as a demonstration of the desired behaviour. The key idea is if a user has to act on devices, the reason is that the user wants to change his environment to improve the current service. A consequence is that user actions are minimized and so functionalities are more relevant. The device can then use this information to self-adapt. However, ambient systems have some particular properties and require functional properties that are challenging for LfD.

Five central questions have been identified that need to be addressed by scientists interested in designing experiments on imitation [14,15]:

- **Who** to imitate: first is the choice of the model (the one to be imitated).
- **When** to imitate: second is determining when imitation has to be done. There are typically two types in literature if whether the imitation is immediately leading to synchronous behaviour or deferred which means that the imitated behaviour might occur even in the absence of the model.
- **What** to imitate: a distinction has to be made between goals, actions and results and which part of the demonstration has to be replicated.
- **How** to imitate: the ***how*** question addresses the problematic of generating an appropriate mapping between the model behaviour and the imitator's one.
- What is a **successful** imitation: one needs to be able to distinguish good imitations from bad imitations. Then, good metrics must be clearly identified.

The application of LfD in the robotic field mainly focuses the "What" and "How" questions [6]. On a previous paper, we position ourselves regarding those questions [16] and discuss specificities of ambient systems.

An ambient system is open, which means that entities can appear and disappear during system activity. This openness property is challenging for LfD, as it does not allow doing assumptions on system's composition. With respect to the openness property, it must be considered that the set of observations $O$ is a priori unknown. One must deal with situations of opulence of data (which induces that some data may be useless in order to learn the task)

and situations of shortage of data where the system functionality is degraded.

To be truly applicable to any kind of system, it has also to be considered that the set of actions A is a priori unknown. Adding a new effector to an ambient system has to be transparent to the learning system. A learner on an ambient system has to adapt its policy to integrate new observations and actions. This adaptation must be dynamic and invisible to users to provide a good quality of service. A *real-time* learning capacity is required for such applications. Thus, the approaches separating learning phase from exploitation (classical *supervised learning* techniques) are not relevant.

Users with their specific and often changing needs are also a source of challenge. In the ambient systems, many different users have day-to-day interactions with the system. Capturing users' needs in such a system cannot be an *ad hoc* process and has to be made along system life-cycle, as users can often change their needs and contradict themselves. The system dynamically adapts its policy to integrate any change in the user's need. An approach based on user profiles appears to be not relevant as user needs could be dynamic and highly specific. The approaches creating a model of the user satisfaction will also suffer of the same lack of adaptivity.

For those reasons, LfD in the ambient systems is relevant for adaptivity to the task to perform (what to do), to the reasons to do it (when to do) and to the users (whom to imitate). The success of imitation is then evaluated by the fact that the user does not have to act on a system any-more. Regarding those specificities, we still need to answer to the last two questions: "What" to learn and "How" to learn.

### 2.4. The "What" and the "How" questions

As interest in imitation learning is as old as the domain of robotics, scientific literature is full of proposals to solve those challenges. Argall et al. [6] categorize techniques for Learning from Demonstration in three classes describing "What" is learnt: *mapping function*, *system model* and *plans*. For each of those categories, we provide a description and an example of application and list the main advantages and disadvantages regarding our application.

- Learning a ***Mapping function***: those techniques calculate a function that associates the current state to action $f() : O \rightarrow a \in A$. Its goal is to reproduce the underlying teacher policy while generalizing in unseen situations. Those techniques use *classification* and/or *regression*. For example, Mitić et al. [17] use a combination of neural network and learning from demonstration to teach visual control of a nonholonomic mobile robot.
  - **pro**: with those approaches, the tutor directly labels the different situations during the demonstration process with the adequate actions to perform. As we intend to use the interactivity of ambient systems as the motor of learning, learning a mapping function appears to be adequate. An example is Chernova et al. [18] who use a set of Gaussian Mixture Models to teach a car how to drive in a busy road. Each Gaussian Mixture Model is incrementally trained with demonstrations gathered from the observation of a human performance driving the virtual car through a keyboard.
  - **cons**: traditional *classification* and *regression* techniques, such as k-Nearest Neighbours (kNN) [19] are separated in two phases: first, gathering the example, and then production of the mapping function. Any change in the user behaviour involves re-performing the whole training process which could be time-greedy and unsatisfying for end-users. More over, those techniques are parametrized, and then need to be tuned for each application.
- Learning ***System model***: those techniques use a state transition model of the world from which they derive a policy. *Reinforcement learning* techniques are a typical case where any state is associated with a reward function. While the usage of reinforcement learning is traditionally limited by the need of large number of environment samples to reach a desirable behaviour, its combination with the LfD paradigm allows to reduce the space search improving its performances. Brys et al. [20] illustrate the gain of performance by comparing performance of reinforcement learning algorithm with or without usage of the LfD paradigm.
  - **pro**: the main advantage of *system models*, and more precisely of using *reinforcement learning* technique is their on-line capacity to learn and their capacity to find optimal behaviours.
  - **cons**: while their on-line capacity to learn is highly desirable in our context, the design of a feedback function is a well-known complex task that has to be hand-crafted for each application. Some approaches, called *inverse reinforcement learning*, propose to use LfD not only to reduce the number of samples but also to learn the feedback function. For example, Knox et al. [21] taught to a robot different kinds of behaviour such as keeping conversational distance or aiming towards the human. They create a predictive model of human reinforcement feedbacks and use this model to increase reinforcement learning algorithms. While they show interesting results in learning a particular task, those approaches are not robust to changes in the task to perform or the system composition and any of those changes involve re-performing the whole learning process.
- Learning ***Plans***: instead of directly mapping states to actions, those approaches represent the desired behaviour as a plan. The *plan* is a sequence of actions that leads from an initial state to the final goal state. For example Mollard et al. [22] propose an approach using plans for robot instruction for assembly tasks. They use low-level demonstrations to learn high-level relational plan of the task. Then, they use a graphical user interface through which the user can interact with the plan to correct both high-level plan or low-level geometrical knowledge of the task.
  - **pro**: plans offer the possibility to draw a readable map of the controller behaviour and then enable intelligibility of the learnt behaviour. Furthermore, they allow to identify goals and to propose different behaviours to reach them.
  - **cons**: the main drawback in using planification algorithm is the complexity of re-planification. The approaches using planification like the one of Mollard et al. [22] propose an interface for the user to assist the system in its re-planification process. However, such techniques prevent their online usage.

Regarding our desired application and state-of-the-art, we would like to propose an approach combining the advantage of mapping functions and system model. Such technique should be able to learn in real time from the observation of user activity without needing to re-perform the whole learning process. Each action from the tutor should be dynamically and autonomously integrated in the learning process allowing our system to be truly self-adaptive.

### 2.5. An extremely sensitive system

There is still a question that has been ignored: who is the learner? This sixth question is often considered as trivial: the robot is the learner. Indeed, the classical approach considers a system as a whole, which means that the system is seen as an omniscient process that exercises a direct control over all of its parts. However, this centralized approach shows some limitations for highly

dynamic and open systems where the observability is strictly local. In a multi-robot application, a robot is a learner but the collective of robots is also learning. The behaviour of the collective is not only the sum of all robots capacities to learn but something greater. Then, the question who is the learner is also the question "where to put the intelligence". Works among the community now consider a decentralized approach where there is no supervisor of the activity [1,4,23]. The "who learns" answer is often "the robot", as the entities are composed of sensors and effectors. Then a robot is controlled by a unique program supervising the activity of all sensors and effectors. The more there are sensors and effectors, the more complex the controller is. The same reasoning can be applied to ambient systems. The more devices are composing the ambient system and the more complex its supervisor has to be.

We argue that this vision is restrictive and propose what we call "Extreme Sensitive Robotic" [16]. The eXtreme Sensitive Robotic (XS Robotic) vision considers each functionality as an autonomous entity. A robot is then composed of many eXtreme Sensitive Functions (XS Functions), globally one for each sensor and effector. An XS Function is sensitive (such as a temperature or ultrasound sensor) or effective (a motor, a led). Each XS Function has to be designed as non final, which involves that the service provided by the functionality has to be able to dynamically self-adapt. Self-observation capacities are the key for sensing variation in the environment and adapting in response. With this approach, the production of a complex behaviour by a robot emerges from the interaction between the different XS Functionalities and the environment. There is thus no difference of design between a single robot, a multi-robot application or a complex ambient system. All those systems are composed of independent functionalities that need to cooperate to perform a global function that is greater than the sum of its parts.

Our proposal is to enable Learning from Demonstration in ambient systems by enabling the EXtreme Sensitive Robotic approach. The next section presents our contribution, ALEX, a multi-agent system for learning by demonstration in ambient robotics. Its conception is based on the Adaptive Multi-Agent System (AMAS) approach and uses recent results on context-aware learning [23,24].

## 3. ALEX: Adaptive Learner by Experiments

The Adaptive Learner by Experiments (ALEX) is a multi-agent system designed to learn from demonstrations to control a robotic device. A robotic device is responsible of a particular functionality (as described in Section 2.2). ALEX has been built upon the Adaptive Multi-Agent System approach (AMAS) [25], which addresses the problems of complex systems with a bottom-up approach where the concept of cooperation acts as the core of self-organization. The theorem of functional adequacy [26] states: "for all functionally adequate systems, there is at least one system with an internal cooperative state that realizes the same function in the same environment". The role of an AMAS is then to automatically detect and repair or anticipate non-cooperative situations by self-organizing to reach a functionally adequate state. ALEX has been designed according to the ADELFE methodology that guides the design of an adaptive system. The ADELFE methodology is based on the well-known software development methodology Rational Unified Process in which some work products specific to the AMAS approach are added [27].

### 3.1. ALEX architecture and general behaviour

An ALEX instance is designed to control a robotic device (basically an effector) by sending actions to it. Those actions are changes of the current state of the robotic device. An ALEX instance is in constant interaction with its environment from which it receives actions from its tutor and a set of sensors values. ALEX observes the current state of all accessible sensors, the action performed by the tutor and in response sends the action to be applied by the controlled robotic device. ALEX is composed of two components, an *Exploitation mechanism* and a set of *Context agents*. Fig. 1 illustrates ALEX architecture.

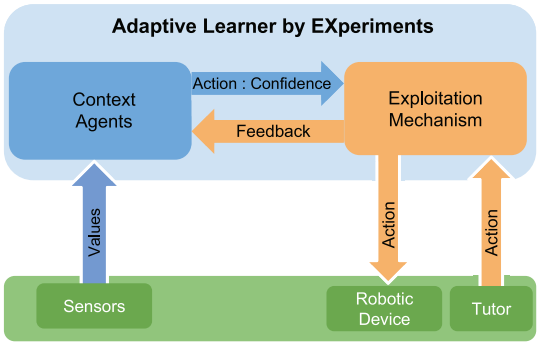


**Fig. 1.** ALEX architecture.

The *Exploitation mechanism* is responsible for sending actions to the robotic device. In order to do so, it receives both the action performed by the tutor and a proposition of action from the set of *Context agents*. By comparing the action realized by the tutor to the proposition made by *Context agents*, the *Exploitation mechanism* can generate a feedback which is sent to the set of *Context agents*. The behaviour of the *Exploitation mechanism* and its feedback generation is listed in the Algorithm 1.

**Algorithm 1** *Exploitation mechanism* pseudo-code

```
repeat
    Action_tutor ← ReceiveTutorAction();
    Action_context ← ReceiveContextAgentAction();
    if Action_tutor = ∅ && Action_context ≠ ∅ then
        sendAction(Action_context);
        sendFeedback(Nothing, Action_context);
    else
        if Action_tutor ≠ ∅ && Action_context = ∅ then
            sendAction(Action_tutor);
            sendFeedback(NoGood, Action_tutor);
        else
            if Action_tutor = Action_context then
                sendAction(Action_context);
                sendFeedback(Good, Action_tutor);
            end if
        end if
    end if
until end
```

The set of *Context agents* is then responsible of making action proposals. The set of *Context agents* is a self-managed memory of previous correlations between the perceived sensors state and the action performed by the tutor. Each *Context agent* has the set of sensors values and the previous *Exploitation mechanism* feedback. *Context agents* must then be created when no *Context agent* proposes the action performed by the tutor and self-adapt whenever they receive a feedback from the *Exploitation mechanism* after an action proposal. At start, the set of *Context agent* is empty. It is then dynamically populated with *Context agent* due to the interaction between ALEX and the tutor. The rest of this section details *Context agents* behaviour, as they are the core of the learning process.

### 3.2. Context-learning with Context agents

Section 2 illustrates that making an exhaustive list of all situations that an ambient system may be faced to is impossible. It is then necessary for the system to dynamically adapt to contexts. The term context refers in this paper to all information external to the activity of an entity that affects its activity. This set of information describes the environment [23]. A system capable of exploiting context information is called "context-aware". ALEX is designed to continually interact with its environment and dynamically learn all the different contexts that can occur. It associates to each context the adequate action to perform. ALEX uses self-observation capacities to dynamically build correlations between the performance of an action and effects of this action on the environment. For thus, an ALEX is composed of a non-finite set of *Context agents* that makes a collective control over an XS functionality. The set of *Context agents* is empty and ALEX dynamically and autonomously creates those *Context agents*. In the next section, we describe the structure and behaviours of *Context agents*. *Context agents* are described by decomposing their behaviours in two kinds. The nominal behaviour is the normal behaviour that the agent performs when the system is in a functionally adequate state. The cooperative behaviour is a subsumption of the nominal behaviour that occurs when the agent needs to self-organize to bring the system towards a functionally adequate state.

### 3.3. Context agents

#### 3.3.1. Context agent nominal behaviour

A *Context agent* associates a low-level context description V (see Section 3.3.2) to a unique action. An action corresponds to maintaining or adjusting the XS functionality. This action is given at the agent creation and never changes (see Section 3.3.6). It can be a high-level command (such as "go Forward", "go Left") or a low-level command ("speed at 42%", "$x = 2.1$") depending on the functionality to control. A *Context agent* receives signals from the environment and uses them to characterize current context. When the observed environment $O$ corresponds to its low-level context description V, it decides to perform its associated action. Otherwise, it does nothing. When a *Context agent* performs an action, the agent is called *selected*.

#### 3.3.2. Low-level context description

Value ranges (named validity ranges) manage the low-level context description. A *Context agent* receives a set of observations $O$ from the environment. Each $o \in O$ is a continuous value such as $o \in [o_{\min}, o_{\max}]$. A validity range $v \in V$ is associated to each observation such as $v_o$ corresponds to the validity range associated to the observation $o$. The set of all validity ranges $V$ is named the validity domain. A validity range is composed of two values, $v_{\min}, v_{\max}$ such as $[v_{\min}, v_{\max}] \subseteq [o_{\min}, o_{\max}]$. A validity range is said **valid** if and only if as $o \in [v_{\min}, v_{\max}]$. Then, a *Context agent* is said **valid** if and only if $\forall v \in V$, $v$ is **valid**. Validity ranges allow the *Context agent* to determine if $O$ is similar to $V$. Adaptive Value Range Trackers (AVRT) manage validity ranges. AVRT is an extension of Adaptive Value Tracker (AVT) [28], a tool that can dynamically find a value from feedbacks greater, lower and good.

#### 3.3.3. The cooperative behaviour

At each time step, only one action can be performed over the robotic device. Thus, each *Context agent* has to determine if its action is the most adequate in the current situation. To be in a functionally adequate state, the system then needs that only one *Context agent* proposes an action. However, situations where more than one *Context agent* proposes an action can occur. Those situations are non-cooperative situations (NCS). Three NCS can occur. The first one (a) occurs when two (or more) agents propose to perform an action (regardless of the nature of the action). The second one (b) occurs if the *Context agent* proposes an action that is not in adequation with the tutor's one. The last one (c) occurs when no *Context agent* proposes an action. To solve these situations, *Context agents* follow a cooperative behaviour to dynamically self-organize.

#### 3.3.4. Confidence

A *Context agent* determines a confidence value that represents the relevance of the action to perform. This confidence value allows *Context agents* to compare their actions. If a *Context agent* wants to perform an action with a confidence value lower than another agent, it has to self-organize its context description to exclude the current situation. Thus, the next time the same context occurs, the *Context agent* will not try performing its action. The confidence value $v$ is ruled by the lambda function: $v_{t+1} = v_t \times (1-\alpha) + F_t \times \alpha$.

$v_{t+1}$ is the confidence value at the step $t+1$. $F_t \in 0|1$ is a feedback value and $\alpha \in [0, 1]$ is a given parameter that models the importance of the feedback. Each time a *Context agent* makes a correct proposal, $F_t = 1$. Otherwise, $F_t = 0$. In our model, $\alpha$ is fixed at 0.8. The more the *Context agent* makes correct proposals, the more its confidence value increases. At creation, $v = 0.5$. The value is then used in the decision process of the *Context agent*. Each *Context agent* can observe the current best confidence value to decide if its action is the most adequate. Confidence allows solving the ambiguity made by the non-cooperative situation (a). A low confidence means that the *Context agent* is often mistaken. We can then introduce a threshold value to detect mistaking Context agents. A *Context agent* with a low confidence level will decide to destroy itself to not disturb the system. In this paper, we set the minimal confidence value at 0.1, meaning that any *Context agent* with a confidence lower than 0.1 will self-destroy.

#### 3.3.5. Adequation with the tutor's action

A tutor can interact at any time with the system by demonstrating the desired behaviour. The demonstration consists in the performance of a particular action $a \in A$. Thus, at any time step, there is a value $a_{tutor} \in A$ corresponding to the tutor's action at this time step. Each *Context agent* can perceive this value thanks to the feedback from the *Exploitation mechanism*. If $a_{tutor} = \emptyset$, the tutor has performed no action. Thus, it is a *Context agent*'s duty to find the most adequate action to perform. However, if $a_{tutor} \neq \emptyset$, *Context agents* can observe tutor's action. If a *Context agent* finds itself valid but proposes a different action than the tutor's one, it has to self-organize to exclude the current situation. Each valid validity range manages its bounds to exclude the current situation. If the set of observations $O$ contains values that are not associated to a validity range (for example, when a new sensor is added on the system), the *Context agent* adds a new validity range to its validity domain corresponding to the new observation. However, if a *Context agent* proposes an adequate action, its confidence value is increased. This allows solving the ambiguity made by the non-cooperative situation (b). The adaptation of a validity range $V$ can result in a situation where $v_{\max} < v_{\min}$. If such a situation occurs, the *Context agent* has a non-coherent structure making it useless. To allow the system to stay in a cooperative internal state, the agent will self-destroy.

#### 3.3.6. "Validable" and context agent creation

The system's functional state requires that at least one *Context agent* is valid at any step (see Section 3.3.3). If at the end of a decision cycle, no *Context agent* proposes to perform an action, the system is in a situation of incompetence. Two mechanisms can solve this situation: extending an already existing *Context agent*

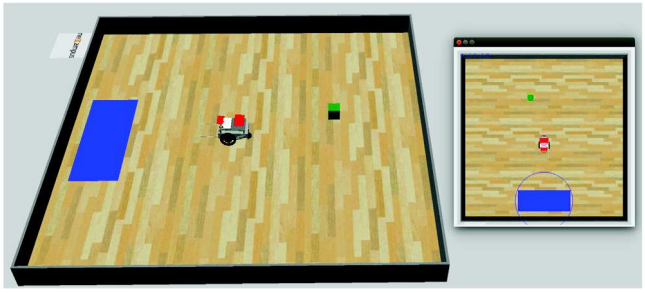

**Fig. 2.** The experiment in Webots. On the left, the rover inside the arena. On the right, the camera detection. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

or creating a new *Context agent* to represent the situation. To anticipate a situation of incompetence, the concept of *validable* is added. A validity range $v_o$ is *validable* if and only if $o \in [v_{\min}, v_{\max}]$ and $o \in [v_{\min} - \delta_{\min}, v_{\max} + \delta_{\max}]$. $\delta_{\min}$ and $\delta_{\max}$ are two values (one for each bound) dynamically managed by the AVTs to control the evolution of each bound. $\delta$ can be interpreted as the next increment of the bound. With the concept *validable*, *Context agents* can propose their action in situation that are not so different to their validity domain. If the *Context agent* proposition is selected, it has to adapt its bound by updating the nearest bound to include the current situation. This mechanism allows *Context agents* to dynamically increase their validity ranges.

The second mechanism occurs when no *Context agent* is *valid* or *validable*. The previously selected *Context agent* (and not valid anymore) can create a new *Context agent* associated with the tutor's action. The created *Context agent* then initializes its validity range around the current position in order to represent the current situation. If $a_{tutor} = \emptyset$, two mechanisms can occur depending on the desired level of autonomy. With a collaborative approach, the system can ask the tutor what is the action to perform. On contrary, with a more autonomous approach, a system will maintain the last known action, considering that the inaction of the tutor corresponds to the maintaining of the current action. Those two mechanisms, the *validable* concept and context creation, allow solving the third non-cooperative situation (c). In the next section, we propose to illustrate ALEX on a set of experiments.

## 4. Experiments

We want to highlight that ALEX presents interesting properties for ambient robotics that differ from the traditional approaches:

- **Genericity**: the learning process is independent of the task to perform and uses no semantic on signals.
- **Openness**: new signals can be dynamically integrated in the decision process.
- **Distributed**: self-observation allows each functionality to be autonomous while remaining cooperative.
- **Real-time**: self-organization is performed without any system downtime. To illustrate these properties, the following experiment has been performed.

In order to do so, we propose to study ALEX behaviour on two different experiments. The first one is the well-known experiment of the Mountain Car problem. The second one is a collecting task involving a 2 wheeled rover. The same implementation of ALEX with no difference of parameter are used in both experiments.

For each experiment, the Tutor (which is either human or virtual) performs a full demonstration of the task to perform. Then, the system acts autonomously under the supervision of the tutor which can act at any time to correct the system behaviour. The results of experiments are obtained after an observation of

**Fig. 3.** Illustration of the mountain car problem: the car (circle) has to try to reach the top of the right hill (right square).

the system in complete autonomy (without tutoring). The first experiment is realized with a virtual tutor which performs a perfect demonstration of the task. The second is performed with a human tutor, which is by nature imperfect.

### 4.1. First experiment: the Mountain Car problem

Initially introduced by Moore [29] and lately defined by [30], the Mountain Car is a well-known toy problem mainly studied in the field of reinforcement learning (Fig. 3). An under-power car situated in a valley must drive up a steep hill. The gravity is too strong for the car's engine so that the car cannot only speed up to climb the hill. The car has to learn to leverage potential energy by driving up to the opposite hill in order to gain enough energy to climb the other hill. As the experiment has been mainly used as a didactic tool to illustrate reinforcement learning algorithms behaviour, the scientific literature is rich of result which can be compared with ALEX.

#### 4.1.1. Problem formalization

The problem is described by [30] as follows:
A two dimensional continuous state space:

- $Velocity \in [-0.07, 0.07]$
- $Position \in [-1.2, 0.6]$.

A one-dimensional discrete action space:

- $motor \in [left, neutral, right]$.

An update function performed at every step:

- $Action \in [-1, 0, 1]$
- $Velocity \leftarrow Velocity + (Action) * 0.001 + \cos(3 * Position) * (-0.0025)$
- $Position \leftarrow Position + Velocity$.

A reward function:

- $reward \leftarrow -1 + height$ with height 0 being the lowest point in the valley.

Initial condition:

- $Position \leftarrow -0.5$
- $Velocity \leftarrow 0.0$.

Termination condition:

- $Position >= 0.6$.

#### 4.1.2. Introducing the tutor

As the mountain car is initially a problem for reinforcement learning techniques, it does not include in its initial description the notion of tutoring. We introduce a virtual tutor which performed the initial demonstration. The virtual tutor acts as an oracle providing at any step the action to perform (Algorithm 2).

This behaviour is considered to be the optimal behaviour to satisfy the user as we are not interested in finding an optimal policy to control the car but to mimic the demonstrated behaviour. With this policy, the task is performed in 125 steps which serves of metric to evaluate likeness of the learnt policy. The virtual tutor strategy is interesting because it is only based on *velocity*. Thus, *position* is a useless data.

**Algorithm 2** Virtual tutor policy

**if** *Velocity* $< 0$ **then**
  **return** $-1$
**else**
  **return** 1
**end if**

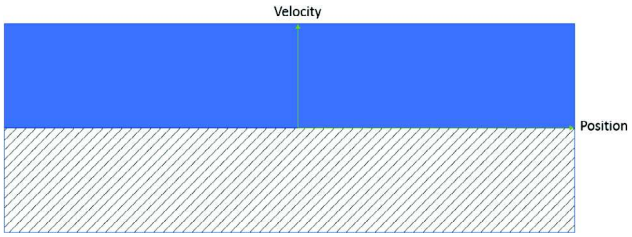


**Fig. 4.** A 2D view of the virtual tutor policy. In abscissa the position of the car, and in ordinate, its velocity. The filled area corresponds to the action 1 and the striped area to the action $-1$.

During the demonstration, the ALEX instance controlling the car receives three signals: the current *Velocity*, the current *Position*, the tutor *Action*.

We define $v_t$ and $p_t$ the velocity and position values at step $t$ and $a_t$ the action performed by the system. Then, the desired ALEX behaviour can be described as the function mapping to any couple of velocity and position the adequate action to perform:

- $ALEX : (v_t, p_t) \rightarrow a_t$.

When the tutor performs an action on ALEX the action is sent to ALEX and *Context agents* use this information to self-adapt. When the tutor performs no action, *Context agents* cooperatively determine which action has to be made. If no *Context agent* proposes to perform an action, ALEX will automatically maintain the last known action.

The two dimensions of the problem allow a 2D projection of the policy. For example, the policy applied by the virtual tutor can be seen in Fig. 4. The projection is composed of two areas. The filled area corresponds to the subspace of the problem where the tutor realizes the action 1 (which means "go forward") and the striped area is the subspace where the tutor realizes the action $-1$ ("go backward"). The virtual tutor policy maps the entire space of the problem, including subspaces which will never be reached due to system dynamics.

The Algorithm 3 describes in pseudo-code the experiment.

**Algorithm 3** Mountain car problem pseudo-code

**Require:** *ALEX*
▷ Initialization
*Velocity* $\leftarrow 0.0$;
*Position* $\leftarrow -0.5$;
▷ While goal is not reached
**while** *Position* $< 0.6$ **do**
▷ Send current data to ALEX
  *ALEX*.*updateObservations*(*Velocity*, *Position*);
  *ALEX*.*newTutorAction*(*VirtualTutorPolicy*());
  *Action* $\leftarrow$ *ALEX*.*getAction*();
▷ Update world state
  *Velocity* $\leftarrow$ *Velocity* $+$ (*Action*) $* 0.001 +$ *cos*$(3 *$ *Position*$) *$ $(-0.0025)$
  *Position* $\leftarrow$ *Position* $+$ *Velocity*
**end while**

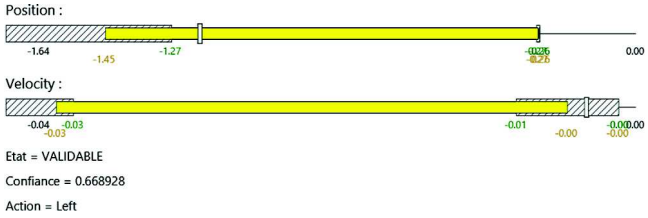


**Fig. 5.** The structure of a *Context agent* involved in the mountain car problem. The agent propose the action *left* (corresponding to action $-1$) when *Position* $\in [-1.45, 0.02276]$ and *Velocity* $\in [-0.03, 0]$.

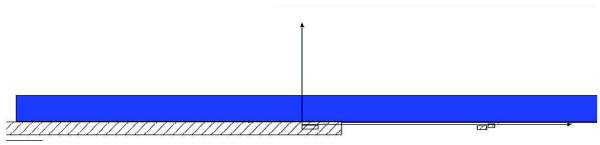

**Fig. 6.** The 2D view of the learnt policy. *Context agents* populates the observed subspace of observation.

#### 4.1.3. Learning from a virtual tutor

In this experiment, the virtual tutor performs a complete demonstration of the hill climbing task. At each step, the virtual tutor provides to ALEX the action to perform. This demonstration is performed in 125 steps. We then realize a second experiment in which previously learned *Context agents* are kept and virtual tutoring is deactivated. ALEX must then autonomously use its knowledge to perform an effective control on the car.

In a unique demonstration, ALEX succeeds to map the observation space with enough *Context agents* in order to be able to perform the task autonomously and as efficiently as the tutor (in 125 steps). A 2D representation of the learnt policy is visible in Fig. 6. We observe that ALEX did not completely map the observation space. Only the space observed during the demonstration is mapped, leaving the unvisited subspace empty of *Context agents*. If we compare the learnt policy (Fig. 6) to the virtual tutor policy (Fig. 4) we observe a similar separation of the space. *Context agents* proposing the action $-1$ occupy the bottom space whereas *Context agents* proposing the action 1 occupy the top space.

If we observe the structure of particular *Context agents* (and example is visible on Fig. 5) we found that validity range are a lot smaller on the *Velocity* than they are on *Position*. As *Position* is a useless data (regarding to the virtual tutor strategy), *Context agents* have learnt to be more sensitive to *Velocity* than *Position*.

#### 4.1.4. Comparison with other approaches

The mountain car problem serves as a toy-problem in the reinforcement learning field [31]. Those reinforcement learning algorithms took interest in finding the optimal policy in least possible steps. Knox et al. [32] propose an approach to reduce the space of research using Learning from Demonstration. Authors propose an algorithm named TAMER which creates a predictive model of human reinforcement feedback and uses this model to increase a reinforcement learning algorithm. Demonstration in TAMER consists in a succession of quantitative feedbacks (for example: $-1$, 0, $+5$) given by a tutor which is only an observer of the system. The experiment they performed on the mountain car problem compared TAMER to the traditional Sarsa [30] algorithm. They show that TAMER reduces the time needed to arrive at a "good" policy while needing more time to find optimality. However, each TAMER agents needs to be shaped for three runs of twenty episodes.

Our own experiment has shown that only one demonstration is required for ALEX to learn a control policy which is as good as the one demonstrated by the tutor. However, in our approach, the tutor

| Camera Observations | Symbol | Domain |
|---|---|---|
| Distance in pixel to the center of the green artefact | $D_g$ | $[-680; 680]$ |
| Angle between the rover front and the green artefact | $A_g$ | $[-2\pi, 2\pi]$ |
| Distance in pixel to the center of the blue artefact | $A_b$ | $[-2\pi, 2\pi]$ |
| Angle between the rover front and the blue artefact | $D_b$ | $[-680; 680]$ |

**Fig. 7.** Camera observations.

is an actor of the system which exhibits a policy that is supposed to be the optimal policy satisfying its needs. This illustrates the advantage of using ALEX for systems where the most important criterion is the likeness of the policy.

### 4.2. Second experiment: collecting artefacts

#### 4.2.1. Description

A two-wheeled rover with no sensor is immersed in a 2 m × 2 m arena composed of a blue area and a green block. An intelligent camera located perpendicularly to the arena at 2 m of its centre can analyse pixels to capture the position of artefacts (the blue area and the green block) relatively to the rover orientation (determined by two red markers on the rover) (Fig. 2). Thus at each time step, the camera can produce four observations on the scene (Fig. 7). To show ALEX openness properties, the camera sends observations only if an artefact is in front of the rover. At each time step, the camera can then produce either zero, two or four observations. This means that some observations can appear and disappear.

A human user performs a direct control over the rover through a 2-joystick gamepad. Each joystick controls the speed of one wheel (left joystick for left wheel and reciprocally). Speed value belongs to $[-100; 100]$ and corresponds to the percentage of the maximum speed to be applied and the sign of the rotation. The user can perform a range of activities in the arena and it is ALEX duty to exploit this interaction to imitate the user performance. The experiment is performed both on real and virtual world. To show both genericity and distributed properties of ALEX, an ALEX is associated to each wheel allowing each wheel to act autonomously. Each wheel is then seen as an autonomous device. The role of an ALEX instance is then to control the wheel speed by correlating the observations from the camera to the actions performed by the user. An ALEX receives at each step of a set of observations from the camera and, if relevant, the action made by the user. One-step of decision occurs every 250 ms. The exact same implementation of ALEX is used on both experiments; only network protocol will differ depending on ALEX controls either a simulated rover or a real rover. The virtual experiment has been developed on *Webots* simulator using the given *Boe-bot* model. The real world implementation has been made with a Boe-bot rover and uses an *Xbee* communication protocol. One of the activities the user can perform on the arena is a collecting task. In order to do so, solid whiskers are added to the rover to allow the rover to capture boxes. Whiskers are not movable and there is no sensor on them. They are just a physical part of the robot. Whenever the boxes are inside the blue area, boxes are moved randomly inside the arena. The Algorithm 4 describes in pseudo-code the experiment.

#### 4.2.2. Context agent creation

The Fig. 8 illustrates the creation of *Context agents* in the two ALEX instances. It shows the trajectory performed by the rover during the first demonstration of the collecting task. In this demonstration, the user takes control over the rover and drives it to collect and transport the box to the blue area. We can observe that ALEX instances create new *Context agents* when the user changes the direction. On contrary, when the user maintains the direction of the rover, no *Context agents* are created. Each user action is observed and each *Context agent* determines if the current

**Algorithm 4** Collecting artefacts pseudo-code

**Require:** $ALEX_{left}$, $ALEX_{right}$
▷ While the experiment is not stopped
**while** *notEnded* **do**
▷ Receive data from Camera
$D_g \leftarrow getDataDg()$;
$A_g \leftarrow getDataAg()$;
$A_b \leftarrow getDataAb()$;
$D_b \leftarrow getDataDb()$;
▷ Send current data to $ALEX_{left}$
$ALEX_{left}.updateObservations(D_g, A_g, A_b, D_b)$;
$ALEX_{left}.newTutorAction(getTutorLeftAction())$;
▷ Send current data to $ALEX_{right}$
$ALEX_{right}.updateObservations(D_g, A_g, A_b, D_b)$;
$ALEX_{right}.newTutorAction(getTutorRightAction())$;
▷ Send new speed command
$leftSpeed \leftarrow ALEX_{left}.getAction()$;
$rightSpeed \leftarrow ALEX_{right}.getAction()$;
$setSpeed(leftSpeed, rightSpeed)$;
**end while**

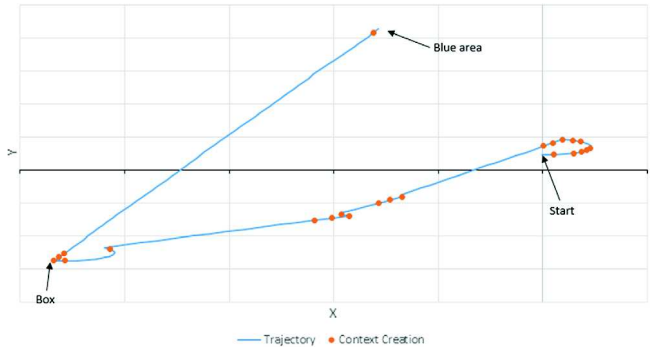


**Fig. 8.** Rover's trajectory and context creation during a demonstration. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

situation belongs to its own context description. If there is no such *Context agent*, a new one is created. This illustrates ALEX capacity to detect new contexts and dynamically self-organize to integrate these new contexts in its decision process.

#### 4.2.3. Self-adaptation of context agents

Fig. 9 shows the structure of a particular *Context agent* at its creation and at the end of a demonstration. Each line corresponds to the structure of a validity range associated to an observation. The filled range corresponds to the valid range whereas the striped area corresponds to the *validable* range. White boxes correspond to the current value of the signal. We observe that each validity range has its own evolution. This evolution is the result of the self-organization process. More precisely, validity ranges associated to the perception of the green block (*GreenA* and *GreenD*) are smaller than the one associated to the perception of the blue area (*BlueA* and *BlueD*). This particular *Context agent* is valid when the block is close to the front of the rover and the rover is in front of the blue area. It is involved in the part of the activity where the rover brings back the block to the blue area.

#### 4.2.4. Performances

To observe the capacity of the system to imitate the user performance, the user realized a 5 min demonstration in which 12 boxes were collected. The number of collected boxes by the user serves as a metric for performance comparison. The system is then let in autonomy and each 5 min the score is computed. During the

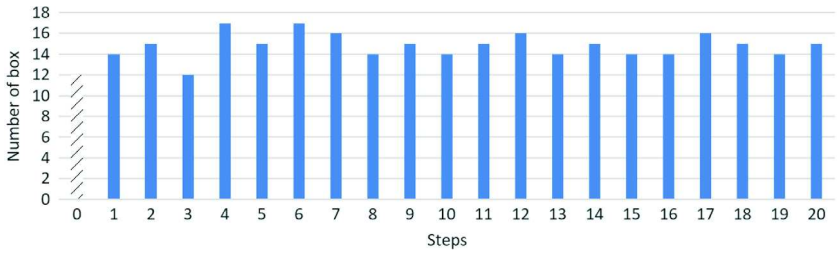


**Fig. 9.** Number of collected boxes each 5 min. The step 0 corresponds to the reference score. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

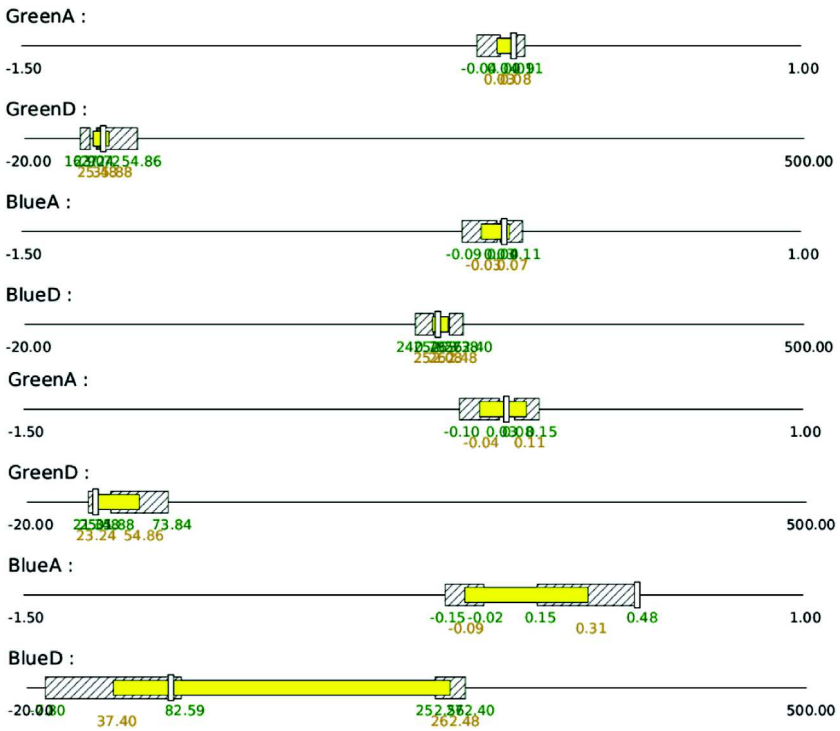


**Fig. 10.** On the top, the validity domain structure at the creation of a *Context agent*. On the bottom, the same *Context agent* at the end of the demonstration.

autonomy phase, *Context agents* have to find the most adequate action. Contrary to the collaborative demonstration phase where the user interacts with the system to teach previously unknown situations, the user never acts on the system. Fig. 9 shows results we obtained. In the worst case, the system performs the task as well as the user does: 12 boxes are collected. However, the number of collected boxes is often better than the user's ones. Two factors influence this result. The first one comes with the randomness of the box movements. In some case, boxes are moved farther away from the blue area and it takes more time to reach and bring back the block. The other one, more interesting, lies in the fact that the user needs more time to take a decision than ALEX does. Moreover, the user can contradict itself. This phenomenon is observable in Fig. 10. At midpoint between the start position and the box position, we can observe a change in the trajectory. This change is in fact a user tele-operation mistake. *Context agents* corresponding to this situation will never be reselected as they correspond to a non-desired action and will then self-destroy. The learnt behaviour is then "filtered" of user mistakes allowing it to perform the task more efficiently.

## 5. Conclusion and future work

This paper deals with the challenges of LfD in ambient robotics. It illustrates that LfD is an interesting approach to learn and generalize recurrent activities. It presents ALEX, a multi-agent system to learn from demonstration in ambient applications. Experiments tend to show the capacity of ALEX to learn from the interaction with its user. While the experiment may appear to be simpler than a realistic situation, the combinatory of the observation space and the randomness of the movement of the artefact illustrate a certain level of complexity, which is a crucial property of ambient systems. Then, the experiment is still illustrative of ALEX behaviour. At last, the study of more realistic situations and limits of the LfD usage are in perspectives. This means concretely that a simple demonstration of the task to perform allows each multi-agent system associated to all the effectors to understand autonomously what the relevant data are in order to mimic collectively what the tutor does, without any central control. Each ALEX creates and self-organizes its *Context agents* to produce collectively a behaviour that is user satisfying. Experiments have shown that two ALEX instances can cooperate without direct interaction. Moreover, it illustrates that the system is able to perform a task more efficiently than the user. This work is a proof of concept that multi-agent context learning is a promising approach to deal with the complexity of ambient systems. We want to consider the use of ALEX technology in more complex problems coming from industrial needs. We have a particular interest for collaborative robotic applications where workers and robots have to work collaboratively. Factories of Future (FoF) are a good illustration of such applications [33]. Evolutions of ALEX algorithm have to include the capacities to filter useless data and

to discover that data are missing. Such processes could be done by adding percept agents associated to each signal which will have the responsibility to learn their utility for *Context agents* and to cooperate with them. Furthermore, on a future approach, we will propose not to imitate user actions, but to learn from the observation of the activity which are the effects on the environment that are desired by the user. The system will then be able to self-discover which are the intentions of the user, and then provides an adequate response to these needs. At last, we want both to formalize our approach and to compare its performances with other well-known learning techniques. Formalization has already led to a proposal of a generic pattern for context learning with *Context agents* [34].

## Acknowledgements

Authors would like to gratefully thank Sogeti High Tech and the ANRT (convention n° 2013/0339) for funding this research. This project is involved in the neOCampus operation (more information at www.irit.fr/neocampus/).

## References

[1] G.A. Kaminka, Autonomous agents research in robotics: A report from the trenches, in: 2012 AAAI Spring Symposium Series, 2012.
[2] J. Chung, Ambient robotics for meaningful life of the elderly, 2014, p. 43.
[3] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cliffs, 1995.
[4] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, H. Duman, Creating an ambient-intelligence environment using embedded agents, IEEE Intell. Syst. 19 (6) (2004) 12–20.
[5] M.J. Kearns, R.E. Schapire, L.M. Sellie, Toward efficient agnostic learning, Mach. Learn. 17 (2–3) (1994) 115–141.
[6] B.D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robot. Auton. Syst. 57 (5) (2009) 469–483.
[7] S. Lefèvre, A. Carvalho, F. Borrelli, Autonomous car following: A learning-based approach, in: Intelligent Vehicles Symposium (IV), 2015 IEEE, IEEE, 2015, pp. 920–926.
[8] N. Vuković, M. Mitić, Z. Miljković, Trajectory learning and reproduction for differential drive mobile robots based on gmm/hmm and dynamic time warping using learning from demonstration framework, Eng. Appl. Artif. Intell. 45 (2015) 388–404.
[9] D. Silver, J.A. Bagnell, A. Stentz, Learning from demonstration for autonomous navigation in complex unstructured terrain, Int. J. Robot. Res. (2010).
[10] M. Power, H. Rafii-Tari, C. Bergeles, V. Vitiello, G.-Z. Yang, A cooperative control framework for haptic guidance of bimanual surgical tasks based on learning from demonstration, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2015, pp. 5330–5337.
[11] A. Billard, S. Calinon, R. Dillmann, S. Schaal, Robot programming by demonstration, in: Springer Handbook of Robotics, Springer, 2008, pp. 1371–1394.
[12] E.A. Billing, T. Hellström, A formalism for learning from demonstration*, Paladyn, J. Behav. Robot. 1 (1) (2010) 1–13.
[13] C.L. Nehaniv, K. Dautenhahn, The correspondance problem, in: Imitation in Animals and Artifacts, MIT press, 2002.
[14] M. Riedmiller, A. Merke, Learning by experience from others—Social learning and imitation in animals and robots, in: Adaptivity and Learning, Springer, 2003, pp. 217–241.
[15] K. Dautenhahn, C.L. Nehaniv, The Agent-Based Perspective on Imitation, MIT Press, 2002.
[16] N. Verstaevel, C. Régis, V. Guivarch, M.-P. Gleizes, F. Robert, Extreme sensitive robotic—A context-aware ubiquitous learning, in: International Conference on Agents and Artificial Intelligence (ICAART 2015), Vol. 1, INSTICC, 2015, pp. 242–248.
[17] M. Mitić, Z. Miljković, Neural network learning from demonstration and epipolar geometry for visual control of a nonholonomic mobile robot, Soft Comput. 18 (5) (2014) 1011–1025.
[18] S. Chernova, M. Veloso, Confidence-based policy learning from demonstration using Gaussian mixture models, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, 2007, p. 233.
[19] J. Saunders, C.L. Nehaniv, K. Dautenhahn, Teaching robots by moulding behavior and scaffolding the environment, in: Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human–Robot Interaction, ACM, 2006, pp. 118–125.
[20] T. Brys, A. Harutyunyan, H.B. Suay, S. Chernova, M.E. Taylor, A. Nowé, Reinforcement learning from demonstration through shaping, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 2015.
[21] W.B. Knox, P. Stone, C. Breazeal, Training a robot via human feedback: A case study, in: Social Robotics, Springer International Publishing, 2013, pp. 460–470.
[22] Y. Mollard, T. Munzer, A. Baisero, M. Toussaint, M. Lopes, Robot programming from demonstration, feedback and transfer, in: Intelligent Robots and Systems (IROS), 2015.
[23] V. Guivarch, V. Camps, A. Péninou, P. Glize, Self-adaptation of a learnt behaviour by detecting and by managing user's implicit contradictions, in: Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Vol. 03, IEEE Computer Society, 2014, pp. 24–31.
[24] J. Boes, F. Migeon, P. Glize, E. Salvy, Model-free optimization of an engine control unit thanks to self-adaptive multi-agent systems, in: International Conference on Embedded Real Time Software and Systems-ERTS$^2$ 2014, 2014, p. 350.
[25] M.-P. Gleizes, Self-adaptive complex systems, in: Multi-Agent Systems, Springer, 2012, pp. 114–128.
[26] D. Capera, J.-P. Georgé, M.-P. Gleizes, P. Glize, The AMAS theory for complex problem solving based on self-organizing cooperative agents, in: Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings, IEEE, 2003, pp. 383–388.
[27] N. Bonjean, W. Mefteh, M.-P. Gleizes, C. Maurel, F. Migeon, Adelfe 2.0, in: Handbook on Agent-Oriented Design Processes, Springer, 2014, pp. 19–63.
[28] S. Lemouzy, V. Camps, P. Glize, Principles and properties of a mas learning algorithm: A comparison with standard learning algorithms applied to implicit feedback assessment, in: 2011 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), Vol. 2, IEEE, 2011, pp. 228–235.
[29] A.W. Moore, Efficient memory-based learning for robot control (Thesis), University of Cambridge, Computer Laboratory, 1990.
[30] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction, Robotica 17 (2) (1999) 229–235.
[31] C. Gatti, The mountain car problem, in: Design of Experiments for Reinforcement Learning, Springer, 2015, pp. 95–109.
[32] W.B. Knox, A.B. Setapen, P. Stone, Reinforcement learning with human feedback in mountain car, in: AAAI Spring Symposium: Help Me Help You: Bridging the Gaps in Human-Agent Collaboration, 2011.
[33] B. Siciliano, F. Caccavale, E. Zwicker, M. Achtelik, N. Mansard, C. Borst, M. Achtelik, N.O. Jepsen, R. Awad, R. Bischoff, Euroc-the challenge initiative for European robotics, in: ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of, VDE, 2014, pp. 1–7.
[34] J. Boes, J. Nigon, N. Verstaevel, M.-P. Gleizes, F. Migeon, The self-adaptive context learning pattern: Overview and proposal, in: International and Interdisciplinary Conference on Modeling and Using Context, CONTEXT, 2015.

**Nicolas Verstaevel** Ph.D. Student—Engineer at Sogeti High Tech.

He is currently Ph.D. student in a partnership project between Sogeti High Tech and the IRIT laboratory.

**Christine Régis** is Assistant professor at Paul Sabatier University.

She works in multi-agent application in robotic application inside the self-adaptive multi-agent system team at IRIT.

**Marie-Pierre Gleizes** Professor at Paul Sabatier University in Toulouse.

She is the team manager of the Self-adaptive multi-agent system at the IRIT laboratory.

**Fabrice Robert** Innovation Manager at Sogeti High Tech.

Leads the R&D team focusing on robotic and artificial intelligence application.